Vertices (or nodes):
A,B,C,D,E,F,G,H,I,J

Edges:
(A,B)  (A,C)
(A,H)  (B,C)
(B,E)  (C,D)
(C,E)  (C,H)
(C,I)  (D,E)
(D,I)  (F,H)
(F,I)  (F,J)
(G,H)  (H,J)
Path:
A series of vertices connected by edges:
Path:
A series of vertices connected by edges:
Path:
A series of vertices connected by edges:
Path:
A series of vertices connected by edges:
// New graph with numvert vertices
graph graph_new(unsigned int numvert);

void graph_free(graph G);

// Number of vertices in the graph
unsigned int graph_size(graph G);

bool graph_hasedge(graph G, vertex v, vertex w);
//@requires v < graph_size(G) && w < graph_size(G);

// Edge can't be in graph!
void graph_addedge(graph G, vertex v, vertex w);
//@requires v < graph_size(G) && w < graph_size(G);
//@requires !graph_hasedge(G, v, w);
// New graph with numvert vertices
graph graph_new(unsigned int numvert);

void graph_free(graph G);

// Number of vertices in the graph
unsigned int graph_size(graph G);

bool graph_hasedge(graph G, vertex v, vertex w);
//@requires v < graph_size(G) && w < graph_size(G);

// Edge can't be in graph!
void graph_addedge(graph G, vertex v, vertex w);
//@requires v < graph_size(G) && w < graph_size(G);
//@requires !graph_hasedge(G, v, w);

struct adjlist_node {
    vertex vert;
    adjlist *next;
};

struct graph_header {
    unsigned int size;
    adjlist **adj;
};